Proceedings of DETC'01
2001 ASME Design Engineering Technical Conferences
September 9-12, 2001, Pittsburgh, Pennsylvania

DETC2001/CIE-21286

COLLABORATIVE MODELING WITH FEATURES

Rafael Bidarra, Eelco van den Berg and Willem F. Bronsvoort

Faculty of Information Technology and Systems
Delft University of Technology
Mekelweg 4, NL-2628 CD Delft, The Netherlands
Email: (R.Bidarra/E.vdBerg/W.F.Bronsvoort)@its.tudelft.nl

ABSTRACT

Collaborative systems are distributed multiple-user systems that are both concurrent and synchronized. An interesting research challenge is to develop a collaborative modeling system that offers all facilities of advanced modeling systems to its users, while at the same time providing them with the necessary coordination mechanisms that guarantee an effective collaboration.

To achieve this, a web-based collaborative feature modeling system, webSPIFF, has been developed. It has a client-server architecture, with an advanced feature modeling system as a basis for the server, providing feature validation, multiple views and sophisticated visualization facilities.

A good distribution of the functionality between the server and the clients has resulted in a well-balanced system. On the one hand, the server offers all the functionality of the original feature modeling system. On the other hand, all desirable interactive modeling functionality is offered by the clients, ranging from display of sophisticated feature model images to interactive model specification facilities.

The architecture of webSPIFF, the distribution of model data, the functionality of the server and the clients, and the communication mechanisms are described. It is shown that a good compromise between interactivity and network load has been achieved, and that indeed advanced modeling with a collaborative system is feasible.

1. INTRODUCTION

In the last decade, research efforts in the areas of solid and feature modeling substantially contributed to the improvement of computer-aided design (CAD) systems. A broad range of advanced modeling facilities is now becoming available in high-end commercial systems, amplified by continuous enhancements in interactive and visualization capabilities, and profiting from the availability of faster and more powerful hardware. Still, these improvements have their counterpart in the increasing size and complexity of such systems. At the same time, a number of research prototypes are pushing the edge to even more advanced modeling facilities. For example, embodiment of richer semantics in feature models and validity maintenance of such models (Bidarra and Bronsvoort 2000), and physically-based modeling techniques (Kagan *et al.* 1999) are among the current research issues.

A common characteristic of most current CAD systems is that they run on powerful workstations or personal computers. Interaction with the system is usually only possible if the user is directly working at the CAD station, although remote interaction is sometimes possible through a high-bandwidth local area network. This situation is no longer satisfactory, as nowadays more and more engineers, often at different locations, are getting involved in the development of products. It would be preferable if a user could remotely browse and manipulate a model, via Internet, as if he were working directly at a powerful CAD station. A web-based system would be ideal for this, as it would facilitate access to all sorts of product information in a uniform, simple and familiar framework.

Even more attractive would be the support of collaborative modeling sessions, in which several geographically distributed members of a development team could work together on the design of a product. Typically, in such collaborative sessions, different participants would be provided with their own, application-specific views on the product model according to the analyses or activities required, e.g. detailed design, manufacturing planning or assembly planning (de Kraker *et al.* 1997; Hoffmann and Joan-Arinyo 1998). In addition, each session participant, as in normal development teams, should be given his own competence and specific session privileges by the system.

So far, only a small number of tools have been developed that somehow support collaborative design activities. For example, tools for collaborative model annotation and visualization via Internet are now becoming available, providing concepts such as shared cameras and telepointers (Autodesk 2000; Parametric 2000; Kaon 2001). However, such tools are primarily focused on inspection, e.g. using simple polygon mesh models, and do not support real modeling activities. In other words, they are valuable assistants for teamwork, but no real CAD systems. Some more recent research is focusing on the possibility of enhancing existing CAD systems with collaborative facilities; see Section 2. To the best of our knowledge, the only commercial system currently offering some collaborative modeling facilities is OneSpace (CoCreate 2000). However, its modeling capabilities are severely constrained by the modeler at the server, SolidDesigner, and by the model format into which it converts all shared models.

The idea of collaborative modeling combines very well with the increasingly popular concept of Application Service Providers (ASP), in which clients remotely access, via Internet, specialized applications running on a server, being billed exclusively for the service time they spend logged on at the ASP server. Such an approach has been identified as a very promising and affordable alternative for distributed CAD teams (Comerford 2000). The first rudimentary commercial CAD ASP has recently been launched by CollabWare (2000).

In order to satisfy all requirements outlined above, it is an interesting research challenge to develop a collaborative modeling system that offers all facilities of advanced feature modeling systems to its users, while at the same time providing them with the necessary coordination mechanisms that guarantee an effective collaboration. Among these mechanisms, solutions have to be provided to the critical problems of concurrency and synchronization that characterize collaborative design environments. This paper presents a new web-based collaborative feature modeling system that is a major step in the right direction. In Section 2, the main research issues of collaborative modeling are surveyed. In Section 3, the architecture of the proposed system is discussed. The structure and functionality of the server and of the clients are described in Sections 4 and 5, respectively. Finally, results and conclusions are presented in Section 6.

2. SURVEY OF COLLABORATIVE MODELING

Collaborative systems can be defined as distributed multiple-user systems that are both concurrent and synchronized. *Concurrency* involves management of different processes trying to simultaneously access and manipulate the same data. *Synchronization* involves propagating evolving data among users of a distributed application, in order to keep their data consistent.

These concepts being in general already rather demanding, their difficulty becomes particularly apparent within a collaborative modeling framework, where the amount of model data that has to be synchronized is typically very large, and the concurrent modeling actions taking place may be very complex. This section briefly surveys collaborative modeling, highlighting the key aspects put forward by recent research, and summarizing the lessons learned from a few prototype systems proposed so far.

2.1. Client-server architecture

The requirements for concurrency and synchronization in a collaborative modeling context lead almost inevitably to the adoption of a *client-server* architecture, in which the server provides the participants in a collaborative modeling session with the indispensable communication, coordination and data consistency tools, in addition to the necessary basic modeling facilities. For a recent survey on client-server architectures, see Lewandowski (1998).

A recurrent problem in client-server systems lies in the conflict between limiting the complexity of the client application and minimizing the network load. In a collaborative modeling context, client complexity is mainly determined by the modeling and interactive facilities implemented at the client, whereas network load is mainly a function of the kind and size of the model data being transferred to/from the clients.

A whole range of compromise solutions can be devised between the two extremes, so-called *thin clients* and *fat clients*. A pure thin-client architecture typically keeps all modeling functionality at the server, which sends an image of its user interface to be displayed at the client. Clicking on the image generates an event, containing the screen coordinates of the interface location the user clicked on. This event is sent to the server, which associates it with an action on a particular widget. Eventually, this action is processed, and an updated image of the resulting user interface is sent back to the client, where it is displayed. This approach requires a continuous information stream between server and clients, and is therefore very expensive in terms of network traffic. The response time would be intolerably high for many model specification actions, thus making it very ineffective to remotely participate in a modeling session.

On the other extreme, a pure fat client offers full local modeling and interaction facilities, maintaining its own local model. Communication with the server is then often required in order to synchronize locally modified model data with the other clients. In a collaborative environment where clients can concurrently modify local model data, preventing data inconsistencies between different clients becomes a crucial problem. In addition, fat clients, to be effective, place on the platform running them the heavy computing power requirements of typical CAD stations.

2.2. Current research prototype systems

Several collaborative modeling prototype systems have recently been described in literature. Some of these systems will be shortly surveyed here, and their shortcomings identified.

CollIDE (Nam and Wright 1998) is a plug-in for the Alias modeling system, enhancing it with some collaborative functionality. Users of CollIDE have private workspaces, where model data can be adjusted independently from other users. In addition, a shared workspace exists containing a global model, which is synchronized between all users participating in a collaborative modeling session. Users can simply copy model data between the private and shared workspaces, in order to create and adjust certain model data locally, and add it to the model in the shared workspace. The architecture of CollIDE poses severe restrictions to crucial collaborative modeling issues. In particular, no special measures have been taken to reduce the amount of data sent between the participants of a collaborative modeling session, resulting in delayed synchronization of the shared workspace and of the users' displays. Also, since each user operates on a separate instance of the modeling system, able to perform modeling operations by itself, concurrency has to be handled by the users themselves in order to keep shared model data consistent.

The ARCADE system (Stork and Jasnoch 1997) defines a *refine-while-discussing* method, where geographically distributed users can work together on a design, interacting with each other in real-time. Every participant uses a separate instance of the ARCADE modeling system, and all ARCADE instances are connected to a session manager via Internet. A message-based approach was chosen, where every change of the product model is converted into a short textual message, which is sent to all other instances of ARCADE through the session manager. ARCADE provides a collaborative environment in which the network load is kept low. This was done by including all modeling functionality in the distributed ARCADE instances, which exchange only textual messages, rather than large sets of polygons. A drawback of this approach, however, is that the user application becomes rather complex, thereby requiring much computational power. In addition, ARCADE provides a primitive concurrency control mechanism, where only one user can edit a particular part at a time.

CSM, the Collaborative Solid Modeling system proposed by Chan *et al.* (1999) is a web-based collaborative modeling system. Within its client-server architecture, the server contains a global model, while every client owns a local copy of this model. When a user has locally modified the model, it is propagated to all other users through the server. Concurrency is managed in two ways: (i) the model can be locked, using token passing, restricting it from being accessed by other users as long as some user is performing a modeling operation; and (ii) functionality can be locked, preventing certain functions from being used by particular users. Clearly, such methods provide a very strict concurrency handling policy. In fact, they turn the clients into several independent modeling systems, just using the same product model alternately. In a truly collaborative modeling system, one expects a higher level of coordination support.

NetFEATURE (Lee *et al.* 1999) claims to be a collaborative, web-based, feature modeling system. A server provides basic functions on a central product model, including creation and deletion of features. On the clients, a local model is available, containing a boundary representation of the product, derived from the server-side central model. The local model is used for real-time display, navigation and interaction. For more advanced operations, the server must be accessed. Updating the local model is done incrementally, which required a rather heavy naming scheme. This scheme severely reduces the modeling functionality of the system, degrading it to a history-based geometric modeling system, instead of a genuine feature modeling system. Furthermore, NetFEATURE uses, just like CSM, very strict concurrency handling methods, thus seriously limiting genuine collaborative modeling.

2.3. Conclusions

Collaborative modeling systems can support engineering teams in coordinating their modeling activities. Instead of an *iterative* process, sending product data back and forth among several team members, designing becomes an *interactive* process, in which several engineers are simultaneously involved to agree on design issues. Collaborative modeling systems typically have a client-server architecture, differing in the distribution of functionality and data between clients and server.

Concurrency control is still a crucial issue in current collaborative environments. If a user is allowed to change a model entity, while another user is already changing the same entity, problems can easily arise concerning consistency. To avoid this situation, a strict concurrency control mechanism can limit access for other users. It depends on the application, whether all entities of the design should be locked or just some of them. If possible, users should be allowed to simultaneously modify different parts of the design, but this could lead to much more complicated concurrency control mechanisms. Also, one should always bear in mind that designing is a constructive activity. Users can therefore be given some responsibility for establishing a good collaboration.

Current systems also often fall short in adequately handling synchronization of model data among distributed clients. Timely updating data over a network is difficult, since there is a certain delay between the moment data is sent and the moment it is received at another node of the network; during this time interval, the latter might try to manipulate data that is not up-to-date. Mechanisms to detect such conflicts should be available, and recovery mechanisms provided. Good locking can also help to avoid such situations, but sometimes it may hinder users' flexibility.

For a collaborative CAD system to be successful, it should provide a good level of interactivity. Users will not be able to design properly if they have to wait a considerable time after every operation. But increasing interactivity by just porting more and more data and functionality to the clients is not a good solution either, as synchronization problems would turn critical again. Furthermore, fat clients are typically platform-dependent applications that require complex installation procedures, and are therefore unsuitable in a web-based context.

In conclusion, a good compromise solution to the difficulties summarized above can be a web-based client-server approach, where the server coordinates the collaborative session, maintains the shared model, and provides all functionality that cannot, or should not, be implemented on the client. The clients perform operations locally as much as possible, and only high-level semantic messages, as well as a limited amount of information necessary for updating the client data, will be sent over the network. This keeps the network load relatively low, while guaranteeing good client interactivity at acceptable response times. An important advantage of such an architecture is that there is only one product model in the system. Clients send their modeling operations to the server, and receive feedback after any modeling operation has been performed on its central feature model, thus avoiding inconsistency between multiple versions of the same model.

3. webSPIFF: A BALANCED SOLUTION

In this section, we discuss the architecture of a system implemented according to the above-mentioned compromise solution: the collaborative feature modeling prototype system webSPIFF.

3.1. Overview of webSPIFF architecture

webSPIFF, the new web-based, collaborative feature modeling system introduced here, has a client-server architecture. As a basis for the server, the SPIFF system developed at Delft University of Technology was chosen, which offers several advanced modeling facilities. First, it offers multiple views on a product model, each view consisting of a feature model with features specific for the application corresponding to the view. The current version of webSPIFF provides two such views: one for design and another for manufacturing planning of parts. In the design view, the feature model consists of both additive (e.g. protrusions) and subtractive (e.g. slots and holes) features. In the manufacturing planning view, the feature model consists of only subtractive features. All views on a product model are kept consistent by feature conversion (de Kraker *et al.* 1997). Second, it offers feature validity maintenance functionality. This can guarantee that only valid feature models, i.e. models that satisfy all specified requirements, are created by a user (Bidarra and Bronsvoort 1999). Third, it offers sophisticated feature model visualization techniques, which visualize much more specific feature information than most other systems do. For example, feature faces that are not on the boundary of the resulting object, such as closure faces of a through slot, can be visualized too (Bronsvoort *et al.* 2001). All these facilities are computationally expensive, and require an advanced product model, including a cellular model with information on all features in all views (Bidarra *et al.* 1998), which uses functionality provided by the ACIS Modeling Kernel (Spatial 2000).

In webSPIFF, some of the functionality of the original SPIFF modeling system, in particular for interaction with feature models, is moved to the clients. However, as soon as real feature modeling computations are required, such as modeling operations, conversion between feature views, feature validity maintenance and feature model visualization, they are executed at the webSPIFF server, on a central product model, and their results are eventually exported back to the clients.

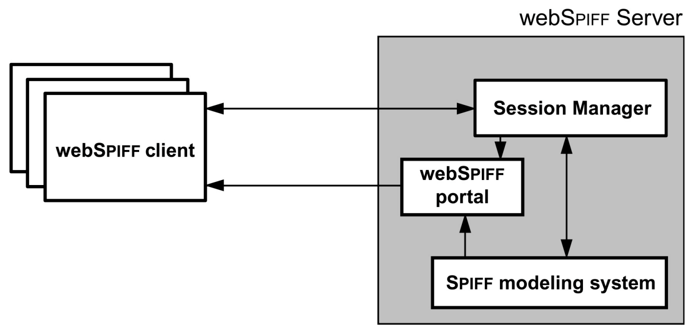

Figure 1 – Architecture of webSPIFF

webSPIFF consists of several components, as depicted in the global architecture diagram of Figure 1. On the server side, two main components can be identified: the SPIFF modeling system, providing all feature modeling functionality; and the Session Manager, providing functionality to start, join, leave and close a modeling session, and to manage all communication between SPIFF and the clients. The webSPIFF portal component provides the initial access to a webSPIFF session for new clients, and includes a web server where model data is made available for download by the clients.

The Session Manager stores information about an ongoing session and its participants. It manages all information streams between webSPIFF clients and the SPIFF process corresponding to the session. Since several session participants can send modeling operations and queries to the webSPIFF server at the same time, concurrency must be handled at the Session Manager. Practically, this means that parallel information streams have to be serialized. The Session Manager has been implemented using the Java programming language (Sun Microsystems 2000).

The clients of webSPIFF make use of standard web browsers. When a new client connects to the webSPIFF portal, a Java applet is loaded, implementing a simple user interface, from which a direct connection with the Session Manager is set up. Different clients can connect from various locations, local through a network or remote via Internet, in order to start or join a modeling session. Using standard web browsers at the clients increases accessibility and platform independence, but limits the complexity of the operations that can be implemented on them. Nevertheless, the main goal of the work described here was to make available to the clients, in an interactive way, as much functionality as possible of the original SPIFF system.

Once connected to the server, the user can join an ongoing collaborative session, or start a new one, by specifying the product model he wants to work on. Also, the desired view on the model, e.g. design or manufacturing planning, has to be specified. Information on the feature model of that view is retrieved from the server, and used to build the client's graphical user interface (GUI), through which the user can start active participation in the modeling session; see Figure 2.

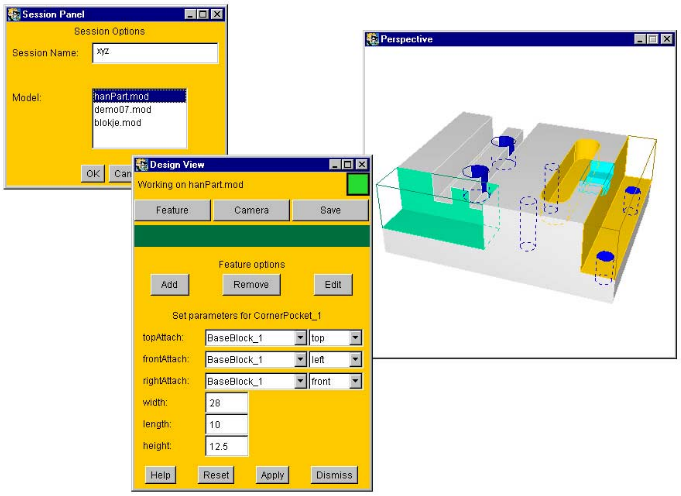

Figure 2 – User interface of a webSPIFF client: the user, working on a design view, is modifying some parameters of the corner pocket at the left side of the model

The bottom line is obviously that clients should be able to specify modeling operations in terms of features and their entities; for example, a feature, to be added to a model, should be attachable to entities of features already in the model (e.g. faces and datums), rather than in terms of faces of the evaluated boundary representation of the product. Among other advantages, this approach avoids the well-known problem of persistent naming of model entities (Bidarra and Bronsvoort 2000). After a feature modeling operation, with all its operands, has been fully specified, the user can confirm the operation. The operation is then sent to the server, where it is checked for validity and scheduled for execution. Notice that this can result in an update of the product model on the server, and thus also of the feature model in the view of each session participant.

In addition to the above functionality, several visualization and interactive facilities of the server have also been ported to the clients. webSPIFF clients provide two ways of visualizing the product model, both making use of so-called *camera* windows. A camera window is a separate window in which a graphical representation of the product model is shown. Each client may create as many cameras as desired. First, a sophisticated feature model image can be displayed. Second, a model can be rendered that supports interactive modification of camera viewing parameters, e.g. rotation and zoom operations. After the desired viewing parameters have been interactively set, they are sent to the server, where a sophisticated feature model image corresponding to the new parameter values is rendered and sent back to the client, where it is displayed. Finally, webSPIFF cameras also provide facilities for interactive specification of modeling operations, e.g. assisting the user in selecting features or feature entities by having them picked on a sophisticated image of the model. The visualization and interactive functionality of webSPIFF cameras is described in detail by van den Berg *et al.* (2000).

To support all these facilities, the webSPIFF clients need to locally dispose of some model data, as described in the next subsection.

3.2. Model data distribution

As explained above, only one central product model is maintained at the server. This feature model includes all canonical shapes, representing individual features in a specific view, and the cellular model. Some model data, however, is also required at the clients. This information is derived by the server from the central model, but it does not make up a real feature model. webSPIFF clients need just enough model information in order to be able to autonomously interact with the feature model, without continuously requesting feedback from the server.

Model data at the clients can be classified into the following three categories:

Textual data

This data is used for specific sets of model information, mostly in list form. The most important are:

- *List of feature classes*: contains the names of all feature classes available in a given view. It is used to fill a GUI list widget when adding a new feature instance, and is requested from the server at client initialization time. This list does not need to be refreshed during a modeling session.

- *List of feature instances*: contains the names of all feature instances in a given view of the model. It is used to fill a GUI list widget when editing or removing an existing feature instance. This list is set upon initialization of the client, and is refreshed after each modeling operation.

- *List of parameter values*: contains the values of all parameters of a given feature instance, in a pre-defined order. It is used to fill various GUI entry widgets when editing the selected feature instance, and is always queried before a feature editing operation.

Graphical data

This comprises the *sophisticated feature model images*, rendered at the SPIFF server in GIF format, and displayed in camera windows at the clients. These images provide very powerful visualizations of a feature model. Many visualization options can be specified. For example, selected features may be visualized with shaded faces, and the rest of the model as a wire frame or with visible lines only. Also, additional feature information, such as closure faces of holes, can be visualized. A separate image is needed for each camera, and it must be updated every time the model or the camera settings are changed. The images are stored in the web server to be downloaded by the respective clients.

Geometric data

This comprises two kinds of models: the visualization model and the selection model.

- *visualization model*: represents the global shape of the product model. The visualization model is generated by SPIFF in VRML format (Ames *et al.* 1997). It is used at the clients for interactively changing the camera viewing parameters. This model is needed because rendering of a smooth sequence of sophisticated feature model images at the server and transmitting it to the clients is unfeasible in real time. All cameras on a particular client use the same local visualization model, but each camera displays it with its own viewing parameters.

- *selection model*: is a collection of three-dimensional objects representing the canonical shapes of all features in a given view of the model. Its purpose is to support the interactive selection of feature faces on the sophisticated feature model image, during the specification of a modeling operation. Again, the selection model is identical for all cameras on a client, each applying its own viewing parameters. The selection model is also generated by SPIFF in VRML format.

Model data on the clients is never modified directly by the clients themselves. Instead, after a modeling operation has been sent to the server and executed, updated model data is sent back to the client. In addition, if the central feature model has been changed, appropriate updated model data is sent to all other session participants as well. This consists of, possibly several, new sophisticated model images, a new visualization model, and an incremental update of the selection model (containing only new and/or modified feature canonical shapes).

Similarly, when a client modifies any camera settings, the corresponding camera operation is sent to the webSPIFF server, which generates a new sophisticated feature model image. Since the feature model remains unaffected by camera operations, the server only needs to send the new sophisticated feature model image back to the client that requested the camera update.

Temporary local inconsistencies can still occur at a client, for example after the execution of a modeling operation. Since sending information from the server to all clients takes some time, for a short period model information on the clients is not up-to-date. Avoiding conflicts arising from this transitory mismatch will be dealt with in Subsection 5.2.

3.3. Data communication

As it becomes clear throughout this section, the various components of webSPIFF have to exchange information at several stages during a modeling session. Communication among them plays therefore an important role in webSPIFF.

webSPIFF clients can specify modeling operations, camera operations, and a variety of queries, and send them to the Session Manager. Communication between the Session Manager and the clients is handled by socket connections. A socket is one end-point of a two-way communication link; there is thus a socket on the client and a socket on the server, with a socket connection set up between them. The Java programming language, used for implementing both the Session Manager and the clients, supports the exchange of complex objects over a socket connection. Messages sent from clients to the Session Manager are simple textual messages, whereas messages sent in the reverse direction are more complex objects, such as arrays of data.

Socket connections also provide a good solution for setting up the communication channel between the Session Manager and the SPIFF modeling system. Textual messages are used to pass commands from the Session Manager to the SPIFF modeling system. The system replies using also textual messages, but several data structures, such as model images and the visualization and selection models, are stored by SPIFF into files at the web server. The Session Manager then notifies the client(s) to download them using the hypertext transfer protocol (HTTP).

4. THE SERVER

As outlined in the previous section, the two main components of the webSPIFF server are the Session Manager and the SPIFF modeling system. The functionality of SPIFF has been summarized in Subsection 3.1; see (Bidarra and Bronsvoort 2000) for a comprehensive overview and additional references. Therefore, this section is focused on the Session Manager, in particular its concurrency management mechanisms.

4.1. The Session Manager

Two important types of processes run on the Session Manager; see Figure 3. First, it maintains a Client Manager for each client, managing the socket connection used for communication with it. The Client Manager receives messages from its client, interprets them, and either processes a message itself, if possible, or propagates it to the SPIFF modeling system. Besides Client Managers, the Session Manager contains one Event Manager for every session, initiated by the Client Manager that opened the session. The Event Manager has an event queue that collects all tasks that have to be passed on to the SPIFF modeling system.

The Client Manager and the Event Manager processes run independently from each other, as so-called separate *Threads* in Java. The main function of a Client Manager is to handle all communication coming from the corresponding client by interpreting messages and taking an appropriate action. A separate Thread is needed for every Client Manager, because within a Thread it is only possible to listen to a single socket at a time.

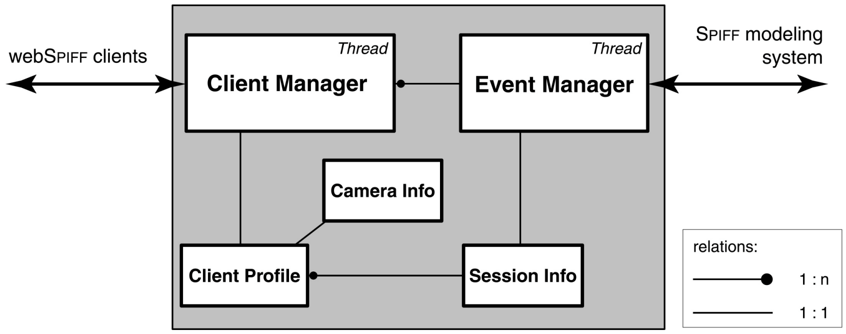

Figure 3 – Structure of the Session Manager

Several types of tasks can be distinguished at the Client Managers. First, session operations have to be handled. These involve starting a session, logging into and out of a session, and also querying for session information. Second, modeling operations can be received that have to be forwarded to the SPIFF modeling system, which, after executing them, returns their result to the Session Manager. Sending these results to all clients is handled by the Event Manager, which retrieves a list of clients from the Session Info record, and starts updating them accordingly. Third, camera operations have also to be forwarded to the SPIFF modeling system. Fourth, queries about the feature model have to be answered. In order to reduce response times, the Session Manager keeps a record of up-to-date model information. In this way, most queries can be directly answered, without involving the SPIFF modeling system. The results of the last two types of tasks must be sent only to the client that issued the request.

When a user starts a new session, a Session Info record is created by the Session Manager, storing all relevant information about the session, for example the name of the session, and a reference to the Event Manager associated with this session.

In the Client Profiles, information about individual clients is stored, for example about the connection between the client and the Session Manager, the modeling view of the user, and a list containing the names of the Cameras the client has opened (Camera Info).

It is the task of the Session Manager to synchronize session participants, by sending them the updated data structures, after a modeling or camera operation has been processed. However, several types of feedback are possible here, depending on the type of operation and whether the operation was successful.

In case of a successful modeling operation, all clients will need updated model data. Messages are then created, notifying them of the appropriate files available for download at the web server (new sophisticated model images, visualization and selection models). Also, additional information on the feature model is included in the messages, such as an updated list of feature instances. Separate messages have to be created per client, since each client typically has different cameras, and therefore requires different feature model images. Of course, the feature model information will also be different for users of different modeling views. Each message is sent to the respective client, using the connection information stored in its Client Profile. Upon receiving its update message, a client can extract and/or download the data in order to update its data structures.

If a modeling operation fails, and the feature model at the server enters an invalid state, the Session Manager takes the role of coordinating the validity recovery efforts. Although several strategies can be devised for such situations, two principles are generally applicable:

- If the validity violation takes place on the modeling view of the client who issued the operation, documentation on the situation, including possible recovery hints generated by SPIFF, is sent to that client, in order to allow him to correct the problem (Bidarra and Bronsvoort 1999). In case there are more session participants working on that view, they are notified as well of the event, although not necessarily allowed to fix the problem.
- If the validity violation takes place on another view than that of the client who issued the operation, it seems appropriate to notify the latter of the event, although documentation and recovery hints are better sent to the client(s) working on the invalid view, who can then take over the task of correcting the problem.

In any case, it seems inevitable to freeze modeling facilities for clients who are not involved at all in the validity violation.

4.2. Concurrency handling

In a distributed multiple-user environment, concurrency must be handled at several stages. If this is not done properly, serious problems can arise, such as inconsistency of data structures, or processes indefinitely waiting for each other, i.e. deadlock. However, not only distributed applications can suffer from concurrency problems. Any application that uses multi-threading must make sure that concurrency involving its resources is managed in a proper way.

Event management

The most delicate case of concurrency occurs with event management at the Session Manager. Communication streams from all clients come together, requesting information to be sent back and modeling operations to be carried out. The Session Manager serializes data that arrives in parallel streams.

As seen in the previous subsection, each client is represented on the Session Manager by an instance of the Client Manager. When two events arrive at the Session Manager at the same moment, only one Client Manager should be able to add an event to the event queue at a time, so the system must determine which Client Manager will be allowed to add the event first. Fortunately, the Java programming language provides useful locking mechanisms for this purpose, the most important being the *synchronized* mechanism. When a class is accessed by a synchronized method, all its methods and data structures are locked, preventing them from being accessed by another process at the same time.

Downloading data using HTTP

Another possible concurrency problem arises when the sophisticated feature model images and VRML files have to be propagated from the web server to the clients. Using HTTP, it is very easy to transfer these files parallel to the existing socket connection between a client and the Session Manager. A problem arises if, for some client, this transfer takes so long that meanwhile the next modeling operation is executed, and the model files stored at the web server are updated before that client completely downloaded their previous version. To overcome this problem, the model data files are buffered at the web server, so that their old version can coexist with the new one: the former for downloading, the latter for writing.

Conflicting modeling operations

webSPIFF encourages users to coordinate their actions, for example using the phone or a chat channel, in order to avoid conflicting operations. To assist in this goal, a *traffic light* icon is displayed on the user interface of every client, informing about the busy state of other clients and of the webSPIFF server. This icon switches from green to yellow when another client starts specifying a modeling operation, and to red when the server starts executing any operation. It has been decided not to implement strict token passing policies, since modeling is considered to be a constructive activity. Additional communication between users will always remain necessary, because it does not make much sense to have several users performing modeling operations simultaneously, without any coordination.

Still, it could happen that two clients decide to simultaneously submit a modeling operation. Consider the following situation: the modeling operation that is handled first by the Session Manager will remove a certain feature, while the second modeling operation will try to edit this same feature. In a single user modeling system, for example SPIFF, this situation could not occur, since operations are always performed serially: after a feature has been removed, it is not possible to subsequently edit it, since it cannot be selected anymore in the user interface. In webSPIFF, however, where operations are performed concurrently, operations can potentially be defined on features that no longer exist. Extra operation validation checks have therefore been added to SPIFF, to ensure that such operations are rejected. A user trying to execute such an operation is notified that his operation is not meaningful anymore.

5. THE CLIENTS

The webSPIFF clients provide the remote user interface to the users of webSPIFF. In order to offer them the same interactive functionality as the SPIFF modeling system does, it is not enough to just replicate the user interface of SPIFF at the clients. As described in the previous sections, webSPIFF clients maintain some data structures with feature model information, and use them to provide various interactive facilities. The interactive functionality of webSPIFF clients was briefly summarized in Subsection 3.1. This section mainly elaborates some communication issues regarding client model data, in particular its synchronization at the clients.

Several components can be identified within a webSPIFF client, as shown in Figure 4. The user interface is the component used for interactively specifying all operations, by means of panels, menus, buttons and list boxes. Three major components can be distinguished, namely the View Panel, the Session Panel and the Cameras. The first two provide plain interfaces with standard widgets, the latter functionality for graphical interaction with the model.

5.1. The Communication Manager

The Communication Manager on the client manages all communication with the Session Manager on the server. It contains a separate process that continuously listens for incoming messages on the socket connected to the Session Manager. Also, outgoing messages bound for the Session Manager are sent via the Communication Manager. Two kinds of messages can be identified here:

a) messages whose response is not awaited, before the client can be operated again; these messages include all modeling operations, such as adding, removing and editing a feature;
b) messages whose response is awaited, suspending all activities on the client; these messages include queries and camera update messages.

When a message of type b) is sent, all activity on the client is suspended until a reply to it is received. While in this state, however, it is still possible that other messages arrive at the client earlier than the expected response. These messages are stored in a queue, and processed after the expected response has been received and processed. An exception is made here for

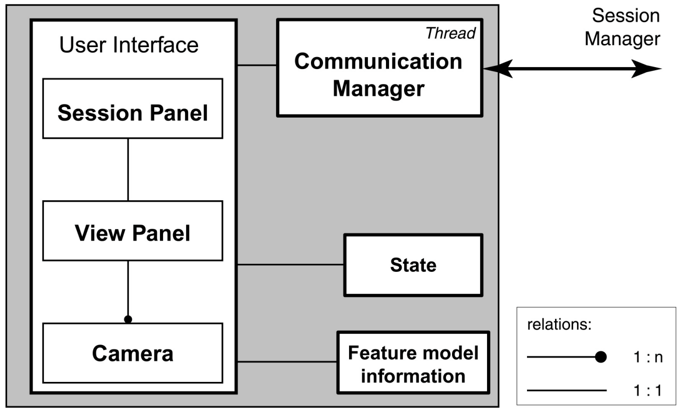

Figure 4 – Architecture of a webSPIFF client

messages concerning the update of state information; see the next subsection.

The reason that camera update messages fall into the second category is that, after a camera operation has been specified and sent to the Session Manager, the interactive functionality cannot be used any more to continue operating webSPIFF. Specifying camera operations is typically done interactively, using the visualization model, whereas specifying modeling operations is typically done using the sophisticated image, in combination with the selection model; see Subsection 3.2. However, after the visualization model has been transformed, both the visualization model and the selection model are inconsistent with the sophisticated feature model image. Therefore, parameters for modeling operations cannot be selected on the sophisticated feature model image until an updated image arrives. For this reason, the client is suspended until the new image has been received.

Whereas messages sent by a client to the Session Manager are merely textual strings, messages received at a client from the Session Manager are more complex, typically containing multiple objects. The first object in all messages is a command string, which is parsed by the Communication Manager to find out the message type, and how it should be handled. In addition, messages can contain requested information on the feature model, such as a list of feature instances, or the names of visualization and selection model files at the web server.

5.2. Synchronization

Synchronization is the process of propagating evolving data from one component of a distributed system to another, in order to keep the information on the components consistent. Before data structures can be updated, it must be made sure that the involved clients are in the right state for processing the update. Two types of information can be distinguished here: (i) updated feature model information, resulting from a modeling or a camera operation, and (ii) updated state information. The order in which these updates are received at the clients is not known in advance, and several scenarios must therefore be handled. Two scenarios are described here.

Updating feature model information

In webSPIFF, the feature model can be modified at any time by one of the users. After such a modification, new feature model information will have to be sent to all clients. At the client side, preparation of a new modeling operation could be underway when the update arrives. It is not convenient to force the user into canceling his operation, since it might well be that the user coordinated his operation with the other users, for example using a communication channel outside webSPIFF, in a way that the update does not have any influence on the modeling operation being specified. For example, a user could be editing a feature, while the update concerns another feature, not having any influence on the edit feature operation that is going on. Having cancelled the operation in this case, would have meant that all the parameters that had been specified so far would have to be specified again. Therefore, the user is allowed to continue specifying the modeling operation, but he is notified of the modeling operation that has been carried out at the server. He can then choose to continue specifying his operation, or to cancel it himself.

Updating state information

Besides updating the feature model information at the client, also state information, such as the traffic light icon, must be kept up to date. The difference with the other data structures is, however, that state information must always be processed as soon as possible. Whereas other messages that arrive unexpectedly can be put into a queue at the Communication Manager, awaiting their processing, the most recent state information must become available to the client immediately, since its purpose is to inform the user of the current state of the modeling session. Therefore, for every incoming message, it is immediately checked whether it is a state update message.

6. RESULTS AND CONCLUSIONS

Current trends in product development demand from CAD systems not only advanced modeling facilities, but also that these be concurrently available to distributed multiple users, supporting effective collaboration sessions among the members of a development team. This paper addresses the new challenges of such requirements. The problems of concurrency and synchronization in a collaborative modeling context can best be handled if a client-server architecture is adopted. Moreover, a web-based approach gives the most advantages, although this requires a careful balance between the conflicting requirements of good client interactivity and low network load.

A new web-based collaborative modeling system, webSPIFF, is presented that provides a solution for many issues involved in collaborative feature modeling systems, including concurrency, synchronization and user interaction. The proposed distribution of functionality between the server and the clients has resulted in a well-balanced web-based system. On the one hand, the full functionality of an advanced feature modeling system is offered by the server. On the other hand, all desirable interactive modeling functionality is offered by the clients, ranging from display of sophisticated images of feature models to interactive selection facilities.

All functionality described in this paper has been implemented in the webSPIFF prototype system. The webSPIFF server runs on a HP B180L Visualize workstation, and its performance during a modeling session is satisfactory. This is mainly due to the well-balanced distribution of functionality and model data between server and clients. The Java-based client application is quite simple. So far, webSPIFF clients running on Unix, Windows and Linux platforms have successfully participated in collaborative sessions. The only requirement at the client side is that it needs to have a Java -enabled web browser, with the Java3D API installed. The webSPIFF portal has a demo version available on Internet for users to experiment with, at www.webSPIFF.org.

Model data files at the web server are small, and therefore download times acceptable. For example, each sophisticated model image takes less than 10 Kbytes, and the selection model less than 5 Kbytes per feature. The visualization model of a moderately complex part is typically smaller than 100 Kbytes. Moreover, the VRML models are downloaded in background, further reducing waiting times at the client. Taking into account the size of these files, it can be questioned whether compressing them before transmission to the clients would further improve system throughput, due to the overhead introduced by the compression and decompression algorithms. It would probably be more effective to use techniques for incremental or progressive transmission of the VRML data; see, for example, Gueziec *et al.* (1999).

Secure transmission and data protection are important issues in any collaborative modeling environment. Although they were not directly considered in the scope of this project, they should receive careful attention in future research.

As Internet technology rapidly improves, faster and better collaboration becomes possible. It can therefore be expected that, although the development of collaborative modeling systems is still at its early stages, such systems will soon play an important role in the product development process.

REFERENCES

Ames, A., Nadeau, D. and Moreland, J. (1997) The VRML 2.0 Sourcebook. Second Edition, John Wiley & Sons, New York

Autodesk (2000) AutoCAD 2000 Online. Autodesk Inc., San Rafael, CA, USA. *http://www.autodesk.com*

van den Berg, E., Bidarra, R. and Bronsvoort, W.F. (2000) Web-based interaction on feature models. Proceedings of the Seventh IFIP WG 5.2 International Workshop on Geometric Modelling: Fundamentals and Applications, University of Parma, Italy, pp. 113–123

Bidarra, R. and Bronsvoort, W.F. (1999) Validity maintenance of semantic feature models. Proceedings of Solid Modeling '99 – Fifth Symposium on Solid Modeling and Applications, Bronsvoort, W.F. and Anderson, D.C (Eds.), ACM Press, NY, pp. 85–96

Bidarra, R. and Bronsvoort, W.F. (2000) Semantic feature modelling. *Computer-Aided Design*, **32**(3): 201–225

Bidarra, R., de Kraker, K.J. and Bronsvoort, W.F. (1998) Representation and management of feature information in a cellular model. *Computer-Aided Design*, **30**(4): 301–313

Bronsvoort, W.F., Bidarra, R. and Noort, A. (2001) Feature model visualization. *Submitted for publication*

Chan, S., Wong, M. and Ng, V. (1999) Collaborative solid modelling on the WWW. Proceedings of the 1999 ACM Symposium on Applied Computing, San Antonio, CA, pp. 598–602

CoCreate (2000) Shared engineering. *http://www.cocreate.com/onespace/documentation/whitepapers/shared_eng.pdf*

CollabWare (2000) An introduction to GS–Design Beta. *https://www.prodeveloper.net/ downloads/whitepaper.pdf*

Comerford, R. (2000) Software, piecewise. *IEEE Spectrum*, **37**(2): 60–61

Gueziec A., Taubin, G., Horn, B. and Lazarus, F. (1999) A framework for streaming geometry in VRML. *IEEE Computer Graphics and Applications*, **19**(2): 68–78

Hoffmann, C.M. and Joan-Arinyo, R. (1998) CAD and the product master model. *Computer-Aided Design* **30**(11): 905–918

Kagan, P., Fischer, A. and Bar-Yoseph, P.Z. (1999) Integrated mechanically-based CAE System. Proceedings of Solid Modeling '99 – Fifth Symposium on Solid Modeling and

Applications, Bronsvoort, W.F. and Anderson, D.C (Eds.), ACM Press, NY, pp. 23–30. Also in: *Computer-Aided Design*, **32**(8/9): 539–552

Kaon (2001) HyperSpace-3DForum. Kaon Interactive Inc., Cambridge, MA, USA. *http://www.kaon.com*

de Kraker, K.J., Dohmen, M. and Bronsvoort, W.F. (1997) Maintaining multiple views in feature modeling. Proceedings of Solid Modeling '97 – Fourth Symposium on Solid Modeling and Applications, Hoffmann, C.M. and Bronsvoort, W.F. (Eds.), ACM Press, NY, pp. 123– 130

Lee J.Y., Kim, H., Han, S.B. and Park, S.B. (1999) Network-centric feature-based modeling. Proceedings of Pacific Graphics '99, Kim, M.-S. and Seidel, H.-P. (Eds.), IEEE Computer Society, CA, pp. 280–289

Lewandowski, S. (1998) Frameworks for component-based client/server computing. *ACM Computing Surveys*, **30**(1): 3–27

Nam, T.J. and Wright, D.K. (1998) CollIDE: A shared 3D workspace for CAD. Proceedings of the 1998 Conference on Network Entities, Leeds. *http://interaction.brunel.ac.uk/~dtpgtjn/neties98/nam.pdf*

Parametric (2000) Pro/ENGINEER 2001*i*. Parametric Technologies Corporation, Waltham, MA, USA. *http://www.ptc.com*

Stork, A. and Jasnoch, U. (1997) A collaborative engineering environment. Proceedings of TeamCAD '97 Workshop on Collaborative Design, Atlanta, GA, pp. 25–33

Spatial (2000) ACIS 3D Modeling Kernel, Version 6.2. Spatial Technology Inc., Boulder, CO, USA. *http://www.spatial.com*

Sun Microsystems (2000) The Sun Java™ Technology Homepage. *http://java.sun.com*